Topic in php.general

_POST & _GET

All 12 messages in topic - view as tree

Greg Nov 11 2002, 12:14 pm show options

Thanks everyone :)
-Greg

"John W. **Holmes**" <holmes072...@charter.net> wrote in message
news:001f01c289b9$bebdf3b0$7c02a8c0@coconut...

- Show quoted text -

Nick Richardson Nov 11 2002, 12:14 pm show options

One thing to watch out for when doing this is that all browsers do not
support the ?foo=blah in the action attribute of the form tag. IE and
Mozilla can handle it fine, but (and what a surprise this is) Netscape
will sometime have some trouble with it (mostly in older versions
(4.x)). But this is the only way that get and post can be combined in
the same form.

As far as using the same php file to generate different forms (some get,
some post)... Yes that's easy. :)

- Show quoted text -

Nick Richardson Nov 11 2002, 12:14 pm show options

yes

- Show quoted text -

John W. Holmes Nov 11 2002, 12:14 pm show options

> Hi-
> Can I use both _GET and _POST in the same php file? Thanks!!
> -Greg
Yes, you can use both in your code, but only one method is going to be
valid when your page is requested by a user. A user can't

request a page
with both a GET and POST request at the same time.
---John **Holmes**

...

.: B I G D O G :. Nov 11 2002, 12:14 pm show options

- Show quoted text -

:.

Rasmus Lerdorf Nov 11 2002, 10:30 pm show options

> ColdFusion developers are more familiar with the URL/form distinction
> than the **get/post** one, as the former are variable scopes. PHP developers
> generally use the **get/post** distinction due to the arrays $_GET and
> $_POST. As I understand it, mod_perl developers do not distinguish the
> two, though I have only heard this from one source (though he is a
> highly respected member of the mod_perl community). I am less certain
> about other communities.

Not that it matters, but the **Get/Post**/Cookie naming in PHP predates the existence of the various $_GET or $HTTP_GET_VARS variables by quite a bit. It was the naming that spawned the variable names, not the other way around. I didn't make up these names. They are right out of the **HTTP** spec. ie. <form action=... method=**GET/POST**>
It seemed obvious and natural to follow this convention in PHP. But yes, the fact that you can have GET variables on a POST request is perhaps somewhat confusing and perhaps the reason why other communities have labelled this data as URL variables. However, I think it is very confusing to call POST variables "form" variables as you can very easily have GET-method forms in which case this "form" data now comes in a URL variable.

Regardless, your explanation was excellent and I hope someone from the documentation teams picks this up and puts it somewhere logical, with your permission of course.

-Rasmus

Chris Shiflett Nov 11 2002, 10:30 pm show options

John W. **Holmes** wrote:
>>Can I use both _GET and _POST in the same php file? Thanks!!

>Yes, you can use both in your code, but only one method is going to be
>valid when your page is requested by a user. A user can't request a page
>with both a GET and POST request at the same time.

It is true that a Web client can only use one method in an **HTTP** request, but this is not as directly related to $_GET and $_POST as it sounds.

There is a sort of naming convention that no one seems to agree on, and that is what to call the different types of data a client can submit. Everyone agrees on cookies, because they are named in a specification, but what do you call variables contained in the query string of a URL? What about variables contained in the content section of a POST request?

As it turns out, different communities use different names. The PHP community, for the most part, uses the term "get variables" to refer to those contained in the query string of a URL. Other communities refer to these as URL variables. The PHP community, again for the most part, uses

the term "post variables" to refer to variables contained in the content section of a POST request. Others sometimes call these form variables.

One important point is that these variables are not defined by the type of request in which they are contained. Consider this **HTTP** request:

```
POST /index.php?foo=bar HTTP/1.1
Host: 127.0.0.1
Content-Type: application/x-www-form-urlencoded
Content-Length: 17

username=shiflett
```

The receiving page, index.php, will have $_GET["foo"] defined as well as $_POST["username"]. So, the answer to the original poster's question (as I interpret the question) is yes.

If you are interested in some of the reasons behind the different choices for naming these variables, read on ...

The use of get variables and post variables originates from the valid values of the method attribute of the HTML form tag. When get is specified, the Web client will submit a GET request to the URL identified in the action attribute. In order to accomodate all of the user's data included in the form, the Web client will include this data in the query string of the URL it requests. In fact, if the following HTML is used ...

```
<form action="/index.php?foo=bar" method="get">
```

... most browsers will basically overwrite the foo variable defined in the URL with all of the user's data submitted in the form. This is a common pitfall. So, because of this behavior of browsers, variables that are included in the URL in this way are often called get variables. It is more of a reference to the method attribute of the HTML form tag than it is a reference to the **HTTP** request method. If a method of post is specified, the Web client will use a POST request and include all of the data in the content section of the request, leaving the URL specified in the action attribute untouched. Thus, variables included in the content section of a POST request are often called post variables.

The reason for the term URL variables is simply because these variables are located within the URL. The use of form variables is a bit less intuitive, and some people passionately disagree with this term when used as an equivalent to post variables. The reason for the disagreement is that many developers refer to form variables as any data submitted in a form, regardless of the method being used. However, those that prefer the term form variables are looking beyond HTML. Consider this HTML form ...

```
<form action="/index.php" method="get">
<input type="hidden" name="foo" value="bar">
<input type="submit">
</form>
```

... and this HTML link ...


```
<a href="/index.php?foo=bar">Click Here</a>
```

When a user submits this form or clicks this link, the request is identical and will look something like this:

```
GET /index.php?foo=bar HTTP/1.1
Host: 127.0.0.1
```

Thus, because there is no difference from a technical perspective, these variables are considered URL variables, regardless of whether they were submitted in an HTML form.

ColdFusion developers are more familiar with the URL/form distinction than the **get/post** one, as the former are variable scopes. PHP developers generally use the **get/post** distinction due to the arrays $_GET and $_POST. As I understand it, mod_perl developers do not distinguish the two, though I have only heard this from one source (though he is a highly respected member of the mod_perl community). I am less certain about other communities

.

That's probably more information than anyone wanted on this topic, but perhaps some people were as curious about this as I once was.

Chris

Chris Shiflett Nov 11 2002, 11:24 pm show options

- Show quoted text -

Yeah, I didn't mean to suggest that actually. I just meant that most PHP developers use the **get/post** distinction due to the PHP variables, as they are likely more influenced by that than by outside resources such as RFCs, etc. Most communities tend to embrace the syntactical properties of their language of choice. In this case, even though the use of "get variables" and "post variables" came before $HTTP_GETS_VARS or the more recent $_GET, your naming decisions have basically shaped this community's perspective.

>I didn't make up these names. They are right out of the **HTTP**
>spec. ie. <form action=... method=**GET/POST**>
>It seemed obvious and natural to follow this convention in PHP.

I prefer PHP's naming convention, but then I cannot claim to be impartial. :-) The use of **get/post** comes directly from the form tag you mention (you mean HTML spec though, right?), and I think this is much more intuitive to developers. The example I gave of a form with method="get" and a link yielding equivalent **HTTP** requests is hidden from developers, so the similarity is not the least bit intuitive.

>But yes, the fact that you can have GET variables on a POST request is perhaps
>somewhat confusing and perhaps the reason why other communities have
>labelled this data as URL variables. However, I think it is very
>confusing to call POST variables "form" variables as you can very easily
>have GET-method forms in which case this "form" data now comes in a URL
>variable.

Yeah, that perfectly illustrates the arguments that each group uses to defend their stance. The URL/form people often cite the fact that URL variables can be present in a POST request, so why call them get variables? The **get/post** people (us) think it is even more confusing to call post variables form variables, since form variables (which we tend to consider any variables submitted in an HTML form) can be get variables.

Anyway, thanks for your insight. Some of these slightly off-topic issues are more interesting than the on-topic ones. :-) Maybe our community needs a historian to collect some of this information for those of us who are interested (or am I the only one?).

Chris

Rasmus Lerdorf Nov 11 2002, 11:24 pm show options

> I prefer PHP's naming convention, but then I cannot claim to be
> impartial. :-) The use of **get/post** comes directly from the form tag you
> mention (you mean HTML spec though, right?)

Well, no, the **HTTP** spec specifies the primitive request methods as being GET, PUT, POST, DELETE, HEAD, OPTIONS and TRACE. It is the request method that you are setting in the HTML form and that method is defined in the **HTTP** spec. But I suppose that is a semantic point.

-Rasmus

Ernest E Vogelsinger Nov 12 2002, 2:18 am show options

WARNING longer post that usual **WARNING** longer post that usual **WARNING**

At 07:54 12.11.2002, Chris Shiflett said:
----------[snip]----------
>Anyway, thanks for your insight. Some of these slightly off-topic issues
>are more interesting than the on-topic ones. :-) Maybe our community
>needs a historian to collect some of this information for those of us
>who are interested (or am I the only one?).

----------[snip]----------

It's slightly more than historical - in fact the behavior of GET and POST is quite clearly laid out in RFC2616 (**http**://ftp.rfc-editor.org/in-notes/rfc2616.txt) which represents the "proposed standard" for **HTTP**/1.1.

To start with, data passed within the URI is called "query", as laid out in rfc2616, 3.2.2:

> 3.2.2 **http** URL
>
> The "**http**" scheme is used to locate network resources via the **HTTP** protocol. This section defines the scheme-specific syntax and semantics for **http** URLs.
>
> http_URL = "**http**:" "//" host [":" port] [abs_path ["?" query]]
> [...]

Si I assume the "correct" way to call our GET variables would be query variables, and the $_GET superglobal should rather be the $_QUERY superglobal...

Rfc2616 also deals with the server side implications of GET and POST methods:

> 9.1.1 Safe Methods
>
> Implementors should be aware that the software represents the user in
> their interactions over the Internet, and should be careful to allow
> the user to be aware of any actions they might take which may have an
> unexpected significance to themselves or others.
>
> In particular, the convention has been established that the GET and
> HEAD methods SHOULD NOT have the significance of taking an action
> other than retrieval. These methods ought to be considered "safe".

This allows user agents to represent other methods, such as POST, PUT
and DELETE, in a special way, so that the user is made aware of the fact that a possibly unsafe action is being requested.

Naturally, it is not possible to ensure that the server does not generate side-effects as a result of performing a GET request; in fact, some dynamic resources consider that a feature. The important distinction here is that the user did not request the side-effects, so therefore cannot be held accountable for them.

In plain english, developers SHOULD use $_GET variables only to _compose_ information, not to take some action to _create_ information. We should always be aware that an URI can always be saved in a link collection ("Favirites"), or used as a link target, as opposed to POSTed data.

9.5 POST

The POST method is used to request that the origin server accept the entity enclosed in the request as a new subordinate of the resource identified by the Request-URI in the Request-Line. POST is designed to allow a uniform method to cover the following functions:

- Annotation of existing resources;

- Posting a message to a bulletin board, newsgroup, mailing list, or similar group of articles;

- Providing a block of data, such as the result of submitting a form, to a data-handling process;

- Extending a database through an append operation.

The actual function performed by the POST method is determined by the
server and is usually dependent on the Request-URI.
[...]

Responses to this method are not cacheable, unless the response includes appropriate Cache-Control or Expires header fields. However,
the 303 (See Other) response can be used to direct the user agent to retrieve a cacheable resource.

[...]

This is as clear as it can be - use POST to perform server-side actions with side effects, like creating a database entity, modifying/deleting a document, etc. Contrary to GET, a POST URI cannot be saved or hyperlinked to, at least not in conjunction with posted data.

Of course the RFC is quite alert of any side effects that might be introduced by us developers ;-> and adds as a note:

13.9 Side Effects of GET and HEAD

Unless the origin server explicitly prohibits the caching of their responses, the application of GET and HEAD methods to any resources SHOULD NOT have side effects that would lead to erroneous behavior if
these responses are taken from a cache. They MAY still have side effects, but a cache is not required to consider such side effects in
its caching decisions. Caches are always expected to observe an

origin server's explicit restrictions on caching.

We note one exception to this rule: since some applications have traditionally used GETs and HEADs with query URLs (those containing a "?" in the rel_path part) to perform operations with significant side effects, caches MUST NOT treat responses to such URIs as fresh unless the server provides an explicit expiration time. This specifically means that responses from **HTTP**/1.0 servers for such URIs SHOULD NOT be taken from a cache. See section 9.1.1 for related information.

It is beyond my knowledge why this rule is seemingly ignored most of the time - maybe most web servers automatically include their default cache mime headers in a response to a GET request, ignoring if it contains a query or not.

Personally I try to use GET (with or without a session identifier) only in application environments that are not security related. When a user is logged in, and this login gets recorded in a SID that is part of a GET request URI, this session can easily ba shared among others while it is active. That wouldn't even be session "hijacking" as it is acceptable to pass an URI along...

But that would be the topic of another thread.

Sorry for the long post, but I believe it is important to have a look at the relevant standards from time to time.

--
>O Ernest E. Vogelsinger
(\) ICQ #13394035
^ http://www.vogelsinger.at/

Charles Wiltgen Nov 12 2002, 9:48 am show options

Ernest E Vogelsinger wrote...
> Sorry for the long post, but I believe it is important to have a look at the
> relevant standards from time to time.

Very informative, thanks!

-- Charles Wiltgen

Chris Shiflett Nov 12 2002, 10:58 pm show options

- Show quoted text -

You should probably read the original discussion. The thread was regarding naming conventions used to refer to variable types, and the reference to historical information was regarding Rasmus's choice of **get/post** for PHP as well as his reasoning.

Also, it should be noted that the **HTTP**/1.1 specification is a draft standard, not a proposed standard. See http://www.rfc-editor.org/rfcxx00.html for the latest status of standards like that.

>To start with, data passed within the URI is called "query"

Not exactly. This is like saying the path in a URL should be called abs_path, since that's what the specification uses. Most everyone uses

the term query string to refer to the section of a URL between the ? and the # (if any). You're the first person I've seen to call it query. :-)

>In plain english, developers SHOULD use $_GET variables only to _compose_
>information, not to take some action to _create_ information. We should
>always be aware that an URI can always be saved in a link collection
>("Favirites"), or used as a link target, as opposed to POSTed data.

>Personally I try to use GET (with or without a session identifier) only in
>application environments that are not security related.

I spoke about the functional differences between GET and POST in a previous email. You might find it helpful:

http://groups.google.com/groups?hl=en&lr=&ie=UTF-8&oe=UTF-8&safe=off&...

Chris

End of messages